MPMM: A MASSIVELY PARALLEL MESOSCALE MODEL

Ian Foster
John Michalakes

Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, Illinois 60439

1. Introduction

Static domain decomposition is a technique that provides a quick path to porting atmospheric models on distributed-memory parallel computers. However, parallel inefficiencies in the form of load imbalances and ill-tuned communication are difficult to correct without complicated and explicit recoding. Reconfiguring the code to run on larger or smaller numbers of processors may require recompiling. Modularity and machine independence may also suffer. If full advantage is to be taken of massively parallel processing (MPP) technology, tools and techniques that allow for dynamic performance tuning and reconfiguration are required.

Program Composition Notation (PCN) is a language and run-time system developed at Argonne and at the California Institute of Technology for expressing parallel programs [2, 3]. It provides an intermediate layer between the application program and the physical processors of a computer. It allows the model to be statically decomposed over a *virtual* machine, but this virtual machine can be mapped and remapped dynamically over the physical computer. Programs are portable to as many machines as PCN itself, modularity is easily preserved, and communication tuning for a particular computer is encapsulated within the PCN run-time system.

In this paper we report on a project at Argonne National Laboratory to parallelize the Penn State/NCAR Mesoscale Model version 5 using a fine grain decomposition dynamically mapped and managed under PCN.

*This work was supported by the Office of Scientific Computing, U.S. Department of Energy, under Contract W-31-109-Eng-38, and was performed in part using the Intel Touchstone Delta System operated by Caltech on behalf of the Concurrent Supercomputing Consortium. Access to this facility was provided by Argonne National Laboratory.

2. The Mesoscale Model and MPP

The Penn State/NCAR Mesoscale Model simulates meso-alpha scale (200–2000 km) and meso-beta scale (20–200 km) atmospheric circulation systems [1, 5]. MM has been developed over a period of twenty years, first at the Pennsylvania State University and more recently also at the National Center for Atmospheric Research. It is used for real-time forecasting and atmospheric research, including climate prediction and storms research. The need for performance has up to now been met by designing and optimizing the model for vector supercomputers, and this approach has been adequate for many current problems. However, while multitasking can increase performance by perhaps an order of magnitude, technological and physical constraints limit the absolute performance that can be attained by conventional supercomputer architectures. In addition, the reliance on custom components makes this approach very expensive. Hence, the MM is costly to run and is near its upper limit of performance on current problems.

Massively parallel processing (MPP) achieves high performance by using, instead of one or a small number of very expensive vector processors, hundreds or thousands of inexpensive microprocessors. By distributing memory locally and connecting processors with scalable communication networks, very large MPP machines may be constructed. Building on recent developments in interconnect technology and microprocessor design, this approach is already competitive with conventional supercomputers and is far from reaching its limits. Hardware architectures capable of scaling to teraflops peak performance have already been announced by Intel and Thinking Machines Corporation, and teraflops computers should be available within five years.

Finite difference codes such as the MM have proven particularly well suited to parallel implementation, because of their regular nearest-neighbor communication pattern. A prototype parallel implementation of version 4 of the MM code executed at nearly one-third the speed of a 1-processor CRAY Y-MP on 12 i860 microprocessors. This prototype used a static (compile-time configured) one-dimensional west/east decomposition of the model grid into equal sized sections. Off-processor data was stored in fixed array extensions and kept up to date by message passing (Figure 1).

To utilize a larger number of processors, the MM grid must be decomposed in a second horizontal dimension. This could be achieved by following a static decomposition strategy, adding array extensions in the second

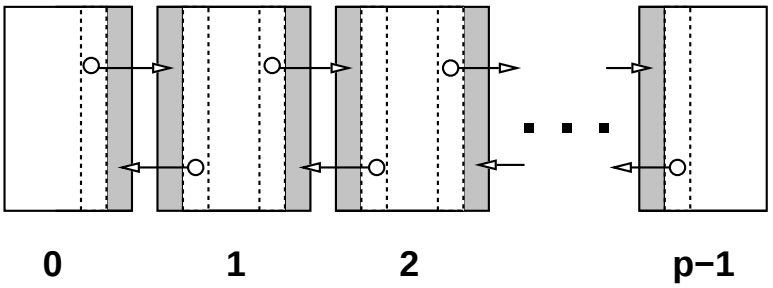

Figure 1: Static decomposition of MM4 grid. Data communication was through extended array "pads" that replicate off-processor memory.

dimension and providing additional structure and communication to account for diagonal data dependencies between grid cells. However, reallocating work to processors for load balancing or machine utilization would difficult, since the static decomposition requires regular rectangular patches. Tuning, in particular overlapping computation and communication, would require complicated explicit recoding that would also introduce machine dependencies. The decision was made to adopt a dynamic rather than a static decomposition when moving from the prototype to a production version of the MM.

2. Massively Parallel MM5

Version 5 of the Penn State/NCAR Mesoscale Model, released in the fall of 1992, incorporates and standardizes a number of features that either were new or that had been added to MM4 for specific applications. Features include a nonhydrostatic option, four-dimensional data assimilation, movable nested grids, and improved physical parameterizations. A pre-release version of the MM5 code was made available for MPMM development in the spring of 1992, and work is continuing. In MPMM, the static decomposition strategy was abandoned in favor of an approach that would support dynamic load balancing and modular implementations of 4D data assimilation, nested grids, and model coupling.

2.1 *Fine-Grained Implementation*

MPMM utilizes a fine-grained horizontal decomposition of the Mesoscale Model domain in which each multicomputer node is allocated some small but *not* statically determined number of columns in the grid. The shape of the processors' allocated region tends toward rectangular (where there are no load imbalances), but columns are able to migrate away from more heavily loaded processors when necessary. The technique that allows for this nonstatic decomposition of the grid is to make a distinction between the logical decomposition from the physical decomposition. The grid is decomposed over a virtual topology of logical PCN processes; the virtual topology is then mapped dynamically onto the physical processors of a particular parallel computer (for example, a mesh of processors as in the Intel Touchstone Delta computer).

The processes are connected by streams over which they communicate needed data to effect horizontal interpolation and finite differencing within the grid. Where communication streams are cut by physical processor boundaries, interprocessor communication is automatically generated by the run time system. Messages over streams between collocated processes are handled as memory references. Moving a process to a different physical processor during model execution does not alter the virtual topology itself, so communication streams "follow" the process to its new physical location (Figure 2). The PCN parallel programming system handles the underlying mechanisms for constructing virtual topologies of processes, mapping them to physical processors, and implementing communication streams automatically.

In addition to the processes representing the model grid, we define a number of global or quasi-global *monitor processes* which implement such global functionality as managing input and output, coordinating load balancing, interfacing with coupled model systems such as a general circulation model, and interfacing with interactive graphical systems. The monitor processes may be mapped to a single physical processor or may themselves be implemented as parallel programs executing on a separate virtual topology of logical nodes.

2.2 *Dynamic Performance Tuning*

Atmospheric models are subject to load imbalances resulting from varying amounts of work over the model grid [6] when decomposed over a set

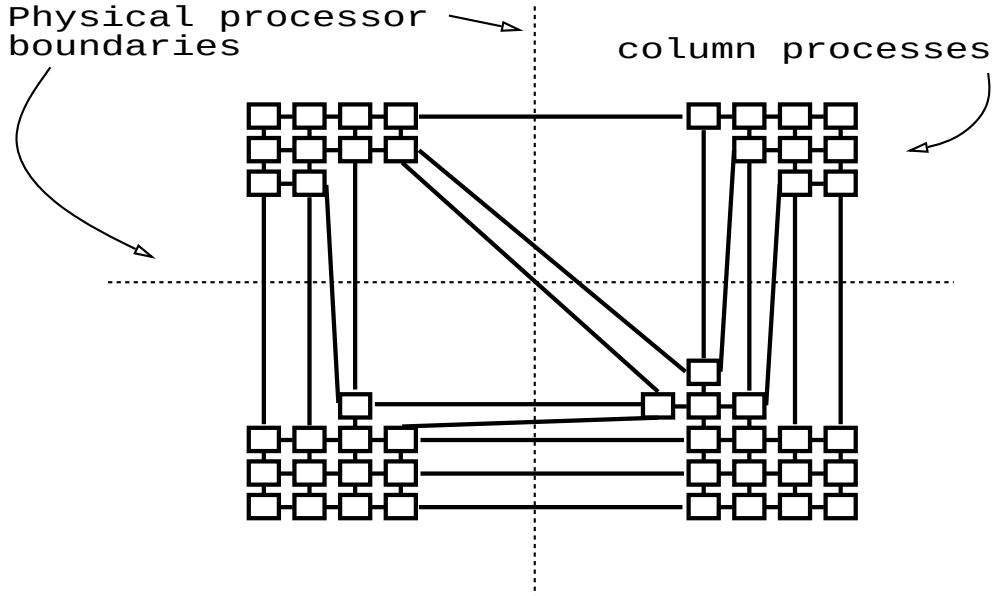

Figure 2: During an MPMM run, column processes in the virtual topology may be migrated away from more heavily loaded physical processors. Communication streams automatically follow under the run-time PCN system implementing the virtual topology.

of distributed memory processors in a multicomputer. MPMM will use dynamic load balancing to maintain parallel efficiency when the amount of work required per grid point is not constant; for example, because of the use of a more elaborate convection scheme in columns containing storm systems or because of dynamically created subgrids. The workload on each processor will be continuously monitored; periodically, imbalances will be corrected by moving vertical grid columns from heavily loaded to lightly loaded processors. This activity is coordinated by a load-balancing monitor process which periodically collects and analyzes load data from each of the processors in the physical topology and instructs column processes in the virtual topology to relocate as needed. Alternative load-balancing algorithms can be substituted without changing other components of the parallel code, allowing a

high degree of customization for load characteristics of a particular modeling application.

The fine-grain decomposition of MPMM provides natural opportunities for the PCN run-time system to overlap computation and communication, effectively hiding communication costs. PCN handles this automatically in the course of scheduling processes that have received their data and are ready to run. For example, a process on the edge of a physical processor's allocation may be blocked waiting for data. During this time, processes on the interior or processes that have already received data execute.

2.3 *Nesting and Coupling*

We intend that MPMM be usable by a broad community of scientists. Critical to this usability will be mechanisms to simplify the implementation of nesting and coupling to other software systems such as other geophysical models but also including interactive graphics packages. We will implement both these capabilities using common mechanisms for transferring data between domains with different resolutions. In essence, a nested grid will be treated as a coupled run of the model at a finer resolution. Each grid will typically be distributed over the entire parallel computer, and appropriate interpolation/averaging routines will be used to transfer data between grids. In the case of coupled models, data transfers may also involve files or potentially parallel versions of other models running on the same computer. We anticipate supporting coupling with BATS and CCM2 initially; other models such as RADM will be considered if required.

The modularity of the design permits the installation of special-purpose monitor processes into the model. Work is currently under way at Argonne to develop a PCN/AVS parallel graphical interface that will allow real-time interactive 2- and 3-dimensional visualization of the model as it executes on a parallel computer. Such an interface could be easily encapsulated within a monitor process and would permit scientists to interactively "explore" the data within their models. Additional modules will support the data movement necessary to implement 4-dimensional data assimilation in a parallel mesoscale model.

3. Implementation

In general, using PCN to parallelize an existing code involves replacing the topmost levels of the original call tree with PCN modules that emulate the original control structure of the program but that also set up and manage parallelism. In MPMM, PCN manages the main loop over time steps and it manages iteration over latitude and longitude, which are now expressed as parallelism. The remaining FORTRAN code has been restructured to operate on individual vertical columns in the grid. That is, the FORTRAN code to compute a complete time step for one cell *i,j* has been transformed into a module. The FORTRAN is restructured with the assumption that all neighboring data is in local memory before the module is called. PCN portions of the code must communicate data between processes as necessary to satisfy this requirement. Therefore, a complete understanding of the horizontal data dependencies in the model is required.

For example, an interpolation of pressure between staggered grids is computed as follows:

```
      hscr1(i,j) = 0.25*(psa(i,j)+psa(i-1,j)+
     $             psa(i,j-1)+psa(i-1,j-1))
```

In addition to the value of PSA at the grid point *i,j*, the value from the west (*i,j-1*), the south, (*i-1,j*), and the southwest (*i-1,j-1*) must be available. Because intermediate results are also required from neighbors, several communications are necessary for each computed time step.

Detailed inspection of the code, assisted by automatic FORTRAN analysis tools developed at Argonne, determined what data was needed by a grid process and when. Figure 3 shows message sizes and sources/destinations for the three communication phases identified for each time step. If the model grid is completely decomposed so that each parallel process represents a single grid cell, the process requires ten bidirectional streams to neighboring processes. This can be reduced to only six if the grid is decomposed so that no fewer than four cells are assigned to each process. This provides the additional benefit that no streams need to pass through a process, avoiding a source of possible message routing contention on some computers.

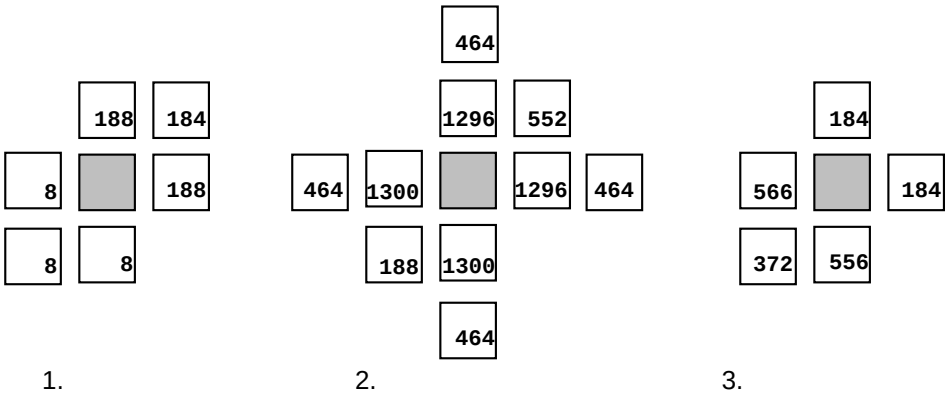

Figure 3: To compute a time step, the shaded grid cell uses data from its neighbors. A process representing the cell must communicate three times with its neighbors. The first communication exchanges data necessary for interpolating pressure between staggered grids and also for decoupling (removing a p^* term) from horizontal wind velocity. The second communication exchanges data needed for second and fourth order differencing for horizontal advection and diffusion. The final step is required for additional interpolation between staggered grids. Communication points and volumes are shown in bytes. Only receives are indicated, but equivalent data must also be sent at each communication step.

4. Performance

At the time of this writing, development work is continuing. This section describes anticipated performanced based on a model of communication and computation in MPMM. The performance model simulates the the cost of computation and the cost to send messages between processors for different mappings of grid processes to physical processors. For a given decomposition, streams are enabled or disabled as necessary to account for interprocessor and intraprocessor communication between processes.

Table 1 shows expected performance for MPMM on the Intel Touchstone Delta computer using the following parameters: The decomposed grid has 40 cells in latitude, 60 in longitude. Grid cells are allocated four to a PCN process, allowing use of the six point communication stencil described earlier. The mapping of processes to simulated processors is as close to regular as possible. The cost of computing a single grid cell is 13 milliseconds. This was based on observed performance of the MM4 prototype. The cost of starting a message is 600 microseconds; the cost for transmitting one byte of the

processors	compute	communicate	total	efficiency
1	31.20	0.00	31.20	1.000
5	6.24	0.34	6.58	0.947
10	3.12	0.22	3.34	0.933
15	2.08	0.24	2.32	0.896
25	1.24	0.17	1.41	0.879
50	0.62	0.11	0.74	0.841
75	0.41	0.10	0.51	0.806
100	0.31	0.08	0.39	0.791
150	0.20	0.06	0.27	0.763
300	0.10	0.04	0.15	0.690
600	0.05	0.02	0.08	0.641

Table 1: Simulated times in seconds for one time step of MPMM on Intel Touchstone Delta computer.

message is 80 nanoseconds [4]. Times shown are for a single time step. As the number of processors increases, parallel efficiency is maintained. Although the amount of work on each processor decreases, so does the number of incoming and outgoing messages.

5. Conclusions

We have described a research and development project intended to develop a massively parallel mesoscale model (MPMM), capable of exploiting both current and future generations of parallel computers. Projected teraflops computers will allow MPMM to achieve performance superior by several orders of magnitude to that currently achievable on conventional supercomputers. In addition, MPMM opens the possibility of using more cost-effective platforms (e.g., networks of multiprocess workstations) for applications that do not require teraflops performance.

MPMM will provide the meteorological community with a cost-effective, high-performance mesoscale model. This in turn will broaden the range and size of problems that can be studied, permitting scientists to consider larger problem domains, longer simulations, finer-resolution grids, and more complex model processes than have previously been possible. In addition, the

parallel algorithms and code developed for MPMM will be directly applicable to projects developing parallel implementations of other, similar models.

References

1. R. Anthes, E. Hsie, and Y. Kuo, Description of the Penn State/NCAR Mesoscale Model Version 4 (MM4). NCAR Technical Note, NCAR/TN-282+STR, 1987.

2. K. M. Chandy and S. Taylor, *An Introduction to Parallel Programming.* Jones and Bartlett, 1991.

3. I. Foster, R. Olson, and S. Tuecke, Productive parallel programming: The PCN approach. *Scientific Programming*, **1**(1), (1992), 51-66.

4. I. Foster, W. Gropp, and R. Stevens, The parallel scalability of the spectral transform method, *Monthly Weather Review*, **120**(5), (May 1992), 835–850.

5. G. Grell, J. Dudhia, and D. Stauffer, MM5: A description of the fifth generation PSU/NCAR Mesoscale Model. Draft NCAR Technical Note, 1992.

6. J. Michalakes, Analysis of workload and load balancing issues in the NCAR Community Climate Model. Argonne National Laboratory Technical Memo ANL/MCS-TM-144, 1991.